Atividade Prática 02 - Estrutura de Dados

Marcos Daniel - 2022069492

3 de setembro de 2023

1 Código-Fonte das funções

A seguir será apresentado o código fonte das funções fibonacci e fatorial em suas versões iterativas e recursivas:

```cpp
// NOTE: This Fibonacci function start with 1, i.e. 1 1 2 3 5 ...
long long fibonacci_recursivo(int n){
    if(n <= 2) return 1;
    // double x;
    //for(int i=0; i<100; i++) x += std::sin(i); // Funcao de consumo de recursos computacionais

    return fibonacci_recursivo(n - 1) + fibonacci_recursivo(n - 2);
}

// NOTE: This Fibonacci function start with 1, i.e. 1 1 2 3 5 ...
long long fibonacci_iterativo(int n){
    if(n <= 2) return 1;
    int a = 1, b = 1, aux;
    for(int i = 0; i < n - 2; i++){
        aux = a + b;
        a = b;
        b = aux;
    }
    return b;
}

long long fatorial_recursivo(int n){
    if(n <= 0) return 1;
    // double x;
    // for(int i=0; i<100; i++) x += std::sin(i); // Funcao de consumo de recursos computacionais

    return n * fatorial_recursivo(n - 1);
}

long long fatorial_iterativo(int n){
    long long res = 1;
    for(int i = 2; i <= n; i++) res *= i;
    return res;
}
```

Listing 1: f_math.cpp

Como é possível observar do código acima, temos as versões iterativas com custo linear de operações, ou seja, $\mathcal{O}(n)$. O código recursivo do fatorial possui também custo linear de operações, sua diferença está na complexidade de espaço que também é linear, tendo em vista que o custo de complexidade de espaço das funções anteriores é constante ($\mathcal{O}(1)$). Já o código recursivo da função de fibonacci possui custo exponencial, $\mathcal{O}(2^n)$.

2 Plano de Experimento

Tendo em vista a ordem de complexidade apresentada acima foi conveniente tomar os seguintes intervalos de testes para a medição de tempo:

- Fatorial iterativo: 10000, 100000, 1000000.
- Fatorial recursivo: 10000, 100000, 1000000.
- Fibonacci iterativo: 10000, 100000, 1000000.

- Fibonacci recursivo: 10, 20, 40.

É importante salientar que embora os algoritmos apresentados realizam corretamente o cálculo para valores pequenos (afinal os resultados tem ordem de crescimento exponencial), para valores grandes como os usados no experimento não são calculados de forma correta e isso se deve por que ocorrem diversas vezes para grandes valores *overflow* do tipo *long long*. Porém, para fins de análise de ordem de complexidade conseguimos fazer uma aproximação. Uma solução mais robusta com a utilização de *BigInt* poderia ser implementada, porém alteraria a forma de tratar os números, logo foi decidido assumir os erros de *overflow*.

3 Relação Tempo de Relógio - Complexidade das Funções

Função	Entrada	Tempo de Relógio
Fibonacci Iterativo	10000	126550 nsec
	100000	443481 nsec
	1000000	3751711 nsec
Fibonacci Recursivo	10	2374 nsec
	20	96379 nsec
	40	2 sec, 413142603 nsec
Fatorial Iterativo	10000	21930 nsec
	100000	196808 nsec
	1000000	1937000 nsec
Fatorial Recursivo	10000	50913 nsec
	100000	234871 nsec
	1000000	2846870 nsec

Podemos, portanto, perceber certa correlação entre os valores obtidos e a complexidade apresentada anteriormente.

3.1 Utilização de recursos e o tempo de relógio

Observou-se ao fazer uma comparação entre esses valores que são correlatos, afinal medem o tempo de forma semelhante. Mas claro com algumas flutuações, isto é, acontecia que as duas medidas variável de maneira semelhante, não igual.

4 Algoritmos Recursivos

Para os algoritmos recursivos foi realizado um experimento no qual foi adicionado um cálculo que consome recursos computacionais no corpo da funções como é descrito no código-fonte no início da atividade. Observe que estão em forma de comentário. A seguir apresento os dados de tempo dessas funções com esse detalhe adicionado.

Função	Entrada	Tempo de Relógio
Fibonacci Recursivo	10	117747 nsec
	20	12637062 nsec
	40	189 sec, 811346961 nsec
Fatorial Recursivo	10000	1904981 nsec
	100000	19080818 nsec
	1000000	189373489 nsec

Antes de apresentar a comparação entre estas duas medidas (GPROF e tempo de relógio), vale salientar que são formas diferentes de medida: enquanto a primeira mede o relógio no início e no final, a segunda realiza o mapeamento de tempo a cada medição periódica realizada. Dessa forma, pode observar diferenças entre elas nos resultados. Além disso, é importante ressaltar que a ferramenta GPROF necessita de tempos maiores porque seu relatório é considerado em segundos, e por isso, na maioria dos casos o GPROF não foi uma ferramenta útil. Tendo aplicação útil apenas no fibonacci recursivo.

4.1 Tempo de Relógio e GPROF

Pela consideração feita anteriormente, vou apresentar os dados somente do caso exceção, ou seja, do fibonacci recursivo. Note que o "0 sec"observado na tabela quer dizer que o processo foi realizada em menos de um segundo.

Função	Entrada	Tempo de Relógio	Tempo GPROF
Fibonacci Recursivo	10	2374 nsec	0 sec
	20	96379 nsec	0 sec
	40	2 sec, 413142603 nsec	2.13 sec

4.2 Tempo de Relógio e GPROF - Uso de função de consumo de recursos computacionais

Após adicionar uma função que consome recursos computacionais, foi observado um aumento significativo no tempo gasto.

Função	Entrada	Tempo de Relógio	Tempo GPROF
Fibonacci Recursivo	10	13100 nsec	0 sec
	20	12637062 nsec	0 sec
	40	189 sec, 811346961 nsec	142.67 sec

5 Notas

- Use o comando *make run* para Compilar e rodar as funções com $n = 5$;
- Use o comando *make gprof* para Compilar e rodar os testes usados acima e gerar os arquivos de gprof. Os arquivos serão gerados na pasta *./tmp/gprof/*.